Plaza: Novel Methods for Offline Bitcoin Commerce, Mesh Payment Relay, and Agent-Native Vending on Open Protocols

Defensive Publication — Technical Whitepaper

Author: Alejandro Diaz, Plaza (plazabtc.com), Oakland, California **Date:** February 22, 2026

Abstract

This paper presents three novel technical contributions to the field of decentralized digital commerce. First, we describe a mesh-relayed ecash commerce system in which Cashu ecash tokens — bearer instruments based on Chaumian blind signatures over Bitcoin — are relayed between merchant point-of-sale terminals via ESP-NOW and BLE Mesh protocols on ESP32 microcontrollers, enabling payment settlement across multiple hops without internet connectivity, with deferred batch settlement to the Lightning Network upon restoration of connectivity. Second, we introduce a beacon-driven commerce discovery pipeline in which BLE beacon advertisements in iBeacon and Eddystone formats broadcast merchant metadata — including product catalogs, pricing, and Cashu mint parameters — enabling customer devices to passively discover merchants, browse offerings, and complete payment via NFC-based Cashu token transfer, all without internet connectivity on either device. Third, we present an agent-native vending architecture in which vending machines operate as autonomous first-class participants on the Nostr protocol, possessing their own cryptographic identity (keypair), publishing inventory as Nostr events, accepting orders via encrypted direct messages, processing payments through Cashu, Lightning, and Nostr zaps, and reporting operational status to their owners via the same open protocol. All three systems are built entirely on open standards and are disclosed here for the purpose of establishing prior art.

---

1. Introduction

1.1 Problem Statement

Contemporary point-of-sale (POS) systems exhibit three fundamental dependencies that

limit their utility and sovereignty:

1. **Constant connectivity requirement.** Systems such as Square, Clover, and Toast require persistent internet connections to authorize transactions. When connectivity is lost — due to infrastructure failure, geographic remoteness, or event density exceeding local network capacity — commerce halts entirely. Square's offline mode permits a 24-hour window of deferred card authorizations but provides no mechanism for verified settlement during the outage.

2. **Centralized payment processing.** All major POS platforms act as payment intermediaries, routing transactions through acquiring banks and card networks. This architecture introduces per-transaction fees (typically 2.5–3.5% plus a fixed per-transaction charge), settlement delays of one to two business days, and the capacity for the intermediary to freeze merchant funds or terminate accounts unilaterally. In the Square ecosystem alone, account freezes constitute the single most frequent merchant complaint.

3. **Platform-controlled identity.** Merchant identity, reputation, customer relationships, and transaction history are stored in proprietary databases controlled by the POS vendor. Merchants cannot port this data to competing platforms without significant friction. Deplatforming — whether by the POS vendor, payment processor, or banking partner — severs the merchant from their accumulated commercial identity.

1.2 Building Blocks

The innovations described in this paper are constructed from four open protocols and one hardware platform:

Nostr (Notes and Other Stuff Transmitted by Relays) is a decentralized protocol for signed event distribution. Each participant holds a Schnorr keypair (secp256k1); events are JSON objects signed by the participant's secret key and distributed via relay servers. Relevant Nostr Improvement Proposals (NIPs) for commerce include NIP-15 (Nostr Marketplace / product listings), NIP-47 (Nostr Wallet Connect), NIP-60 (Cashu Wallet), NIP-61 (Nutzaps), NIP-69 (Peer-to-Peer Order Events), and NIP-99 (Classified Listings).

Lightning Network is a Layer 2 payment channel network on Bitcoin enabling near-instant, low-fee transactions. Payments are routed through a network of bidirectional payment channels using Hash Time-Locked Contracts (HTLCs). For merchant applications, Lightning invoices (BOLT 11) provide a standard mechanism for requesting payment.

Cashu is an ecash protocol implementing David Chaum's blind signature scheme over Bitcoin. A mint issues bearer tokens ("proofs") in exchange for Lightning payments. Each proof is a blind-signed commitment from the mint; the bearer can transfer it to any counterparty without further interaction with the mint. The counterparty later redeems the

token at the mint. Cashu tokens are serialized in a compact format (cashuA or cashuB prefix followed by base64-encoded JSON) and are natively offline-transferable since the mint interaction occurs only at issuance and redemption, not at transfer.

BLE Mesh and ESP-NOW are wireless communication protocols supported natively by the Espressif ESP32 family of microcontrollers. ESP-NOW is a connectionless protocol operating on the 2.4 GHz band that transmits packets of up to 250 bytes at distances up to 480 meters line-of-sight without requiring a WiFi access point or router. BLE Mesh is a Bluetooth Low Energy mesh networking standard (Bluetooth SIG Mesh Profile) enabling many-to-many device communication with message relay and managed flooding.

ESP32 is a dual-core microcontroller with integrated WiFi (802.11 b/g/n) and Bluetooth 5.0 (including BLE). The ESP32-S3 variant provides 8 MB flash, 2 MB PSRAM, and hardware cryptographic acceleration. It is the hardware platform for all terminal and vending implementations described herein.

---

2. Background & Related Work

2.1 Existing POS and Payment Systems

Square (Block, Inc.) provides a vertically integrated POS ecosystem with hardware readers ($59–$899), mobile and tablet software, and centralized payment processing. Square charges 2.6% + $0.15 per in-person transaction and settles funds in one to two business days. In 2025, Square added Lightning Network support with a $600 per-transaction limit, custodial Bitcoin handling, and a temporary 0% fee promotional period through 2027. Square's offline mode stores card data locally for up to 24 hours but carries chargeback risk and provides no cryptographic settlement guarantee.

BTCPay Server is an open-source, self-hosted payment processor supporting Bitcoin on-chain and Lightning invoicing via its Greenfield API. BTCPay provides no dedicated mobile application, no hardware terminal, no mesh networking capability, and no integration with Nostr identity or Cashu ecash.

Strike provides a Lightning-based payment API enabling merchants to receive Bitcoin payments with optional instant USD conversion. Strike operates as a centralized, licensed money transmitter and offers no POS application, no hardware, and no offline payment capability.

2.2 Mesh Networking Precedents

Bitchat is an open-source BLE mesh messaging project developed by Jack Dorsey and released in July 2025. Bitchat demonstrated that BLE mesh networks can scale rapidly —

reaching 10,000 users within 48 hours of its TestFlight beta release. Bitchat transmits text messages over BLE mesh and does not implement payment functionality, ecash relay, or commerce discovery.

goTenna and similar mesh networking devices provide off-grid text communication but do not integrate with any payment protocol.

2.3 Cashu Protocol

The Cashu protocol, specified at https://github.com/cashubtc/nuts, defines a set of Notation Units (NUTs) governing mint operations. Key NUTs relevant to this work:

- **NUT-00:** Token serialization format. A Cashu token is a JSON object containing `token` (array of `TokenEntry` objects, each containing a `mint` URL and `proofs` array) serialized as `cashuA` + base64url encoding. The v2 format (`cashuB`) adds additional metadata fields.
- **NUT-01:** Mint keypairs. The mint publishes a set of public keys, one per denomination (1, 2, 4, 8, 16, ... sats). Each key signs blinded secrets for that denomination.
- **NUT-02:** Keysets. A keyset is an identified set of mint keys, allowing key rotation.
- **NUT-03:** Swap. Token holders can swap proofs for new proofs at the mint without revealing linkage (unlinkability property).
- **NUT-04:** Mint tokens (issue new proofs in exchange for Lightning payment).
- **NUT-05:** Melt tokens (redeem proofs by having the mint pay a Lightning invoice).
- **NUT-07:** Token state check. Allows querying the mint to determine whether a proof has been spent.
- **NUT-10:** Spending conditions. Proofs can carry spending conditions (P2PK, HTLC) in their secret field.
- **NUT-11:** Pay-to-Public-Key (P2PK). Tokens locked to a specific public key.
- **NUT-13:** Deterministic secrets. Derives secrets deterministically from a seed.

2.4 Relevant Nostr Standards

- **NIP-01:** Basic event structure. Events contain `id` , `pubkey` , `created_at` , `kind` , `tags` , `content` , `sig` .
- **NIP-04 / NIP-44:** Encrypted direct messages (legacy and versioned, respectively).
- **NIP-15:** Nostr Marketplace. Defines event kinds for stalls (kind 30017) and products

(kind 30018).

- **NIP-47:** Nostr Wallet Connect. Defines a protocol for remote wallet control via Nostr events.
- **NIP-60:** Cashu Wallet. Defines event kinds for storing Cashu wallet state on Nostr relays.
- **NIP-61:** Nutzaps. Cashu-based micropayment tips via Nostr.
- **NIP-69:** Peer-to-Peer Order Events. Defines event kinds for direct merchant-customer order flow.
- **NIP-87:** Mint and federation discovery on Nostr.
- **NIP-99:** Classified Listings.

2.5 Novelty Statement

No existing system combines (a) mesh-relayed ecash token forwarding between terminals for deferred Lightning settlement — including autonomous supply chain settlement with cryptographic audit trails, (b) BLE beacon-driven offline merchant discovery with NFC ecash payment completion, or (c) autonomous vending machines operating as first-class Nostr protocol participants with their own cryptographic identity. Each of these three innovations, and their combined integration, is novel and is disclosed here as prior art.

---

3. Innovation I: Mesh-Relayed Ecash Commerce

3.1 Overview

This section describes a system in which Cashu ecash tokens received by a merchant terminal via NFC are stored locally, relayed to other terminals via ESP-NOW or BLE Mesh, and ultimately settled to the Lightning Network when any terminal in the mesh regains internet connectivity. The system enables multi-hop token relay, allowing a payment to traverse an arbitrary number of intermediate terminals before reaching an internet-connected settlement node.

3.2 System Architecture

The system comprises:

1. **Customer device:** A smartphone running a Cashu-compatible wallet (e.g., Minibits, eNuts, Cashu.me) with NFC capability.

2. **Merchant terminals:** ESP32-based devices equipped with PN532 NFC readers, ILI9341 TFT touchscreens, and BLE/ESP-NOW radios. Each terminal maintains local storage (8 MB flash, SQLite over LittleFS — see §3.5) for token caching and mint keyset data.
3. **Supplier terminals (optional):** Additional Plaza terminals operated by upstream suppliers (distributors, wholesalers, producers), participating in the same ESP-NOW/BLE mesh network. Supplier terminals receive accumulated Cashu tokens from merchant terminals via mesh relay or Lightning bridge, enabling end-to-end supply chain settlement without banking infrastructure (§3.9).
4. **Cashu mint(s):** One or more ecash mints whose public keysets are cached on each terminal.
5. **Lightning Network:** Settlement layer for redemption of accumulated Cashu tokens.

3.3 Token Reception via NFC

When a customer initiates payment, the following sequence occurs:

1. The merchant terminal displays the requested payment amount on its ILI9341 screen and enters NFC listen mode on the PN532 reader.
2. The customer's wallet application constructs a Cashu token (per NUT-00) with proofs summing to the requested amount, selects proofs from a mint whose keyset the terminal recognizes, and writes the serialized token to the NFC Data Exchange Format (NDEF) payload.
3. The customer taps their phone to the terminal's NFC reader. The PN532 reads the NDEF record containing the Cashu token.
4. The terminal parses the token, extracts the `mint` URL and `proofs` array.

3.4 Local Token Verification

Before confirming payment, the terminal performs local verification:

1. **Keyset verification:** The terminal checks that the token's keyset ID matches a keyset in its local cache. Each terminal stores the public keys for configured mints, fetched during setup and periodically refreshed when online (per NUT-01 and NUT-02). The keyset includes the mint's public keys for each denomination.
2. **Signature verification:** For each proof in the token, the terminal verifies the Chaum blind signature. Specifically, the terminal:
    - Extracts the proof's `C` value (the signed blinded message, a curve point).

- Extracts the proof's `amount` field to identify the corresponding mint public key `K_amount`.
- Computes `Y = hash_to_curve(secret)` where `secret` is the proof's secret string, using the hash-to-curve algorithm specified in NUT-00 (a deterministic mapping from the secret string to a point on secp256k1).
- Verifies that `C - r * G == K_amount * Y` (unblinding), where `r` is not available to the terminal for bearer tokens. For standard bearer proofs (not P2PK-locked), the terminal can verify the DLEQ proof (NUT-12) if attached by the mint. The DLEQ proof demonstrates that `log_G(K_amount) == log_Y(C)` — i.e., that the same discrete logarithm (the mint's private key for that denomination) relates `K_amount` to the generator `G` and `C` to the point `Y = hash_to_curve(secret)`. This proves the mint correctly signed the token without the verifier knowing the blinding factor `r`.
- Alternatively, when DLEQ proofs are not available, the terminal accepts the token provisionally, relying on the mint's double-spend check (NUT-07) at redemption.

3. **Token amount verification:** The terminal sums the proof denominations and confirms they meet or exceed the requested payment amount.
4. **Token TTL check:** Each locally accepted token is assigned a timestamp. Tokens older than a configurable TTL (default: 72 hours) are flagged for priority settlement and excluded from further relay.

Upon successful verification, the terminal displays a confirmation on the touchscreen and emits an audible acknowledgment.

3.5 Local Token Storage

Verified tokens are stored in a local SQLite database on the ESP32's flash filesystem. The ESP32 runs SQLite over a LittleFS partition — a configuration that functions correctly because LittleFS provides a POSIX-like file interface with wear leveling, and SQLite's VFS abstraction layer maps standard file operations onto LittleFS without modification. This is a tested configuration in ESP-IDF v5.x environments. Each stored token record includes:

```
{
  "token_id": "<SHA-256 hash of serialized token>",
  "mint_url": "https://mint.example.com",
  "keyset_id": "00a1b2c3d4e5",
  "proofs": [ ... ],
  "amount_sats": 500,
  "received_at": 1740249600,
  "received_from": "nfc",
  "relay_count": 0,
```

```
  "relay_path": [],
  "settled": false,
  "ttl_expires": 1740508800
}
```

3.6 Mesh Relay Protocol

When a terminal holds unsettled tokens and lacks internet connectivity, it participates in the mesh relay protocol:

3.6.1 ESP-NOW Relay

ESP-NOW is the primary relay mechanism due to its simplicity and range.

Discovery phase: Each terminal periodically broadcasts an ESP-NOW beacon frame (every 10 seconds) on a predetermined channel (default: channel 1). The beacon frame structure:

```
Byte 0:       Protocol version (0x01)
Bytes 1-6:    Terminal MAC address
Byte 7:       Flags [bit 0: has_internet, bit 1: has_unsettled_tokens, bit 2: acc
Bytes 8-11:   Unsettled token count (uint32, little-endian)
Bytes 12-15:  Free storage bytes (uint32, little-endian)
Bytes 16-47:  Supported mint keyset IDs (packed, 4 bytes each, up to 8 mints)
Bytes 48-49:  Checksum (CRC-16)
```

Relay initiation: When a token-holding terminal (the **sender**) detects a beacon from a terminal with `has_internet=1` or with a shorter path to an internet-connected node, it initiates token relay:

1. Sender transmits a `RELAY_REQUEST` frame to the target terminal:

   ```
   Byte 0:       Message type (0x10 = RELAY_REQUEST)
   Bytes 1-6:    Sender MAC
   Bytes 7-10:   Number of tokens to relay (uint32)
   Bytes 11-14:  Total sats value (uint32)
   Bytes 15-18:  Mint keyset ID of tokens
   Bytes 19-20:  Checksum
   ```

2. Receiver responds with `RELAY_ACCEPT` (0x11) or `RELAY_REJECT` (0x12). Rejection reasons: insufficient storage, unknown mint keyset, rate limiting.

3. Upon acceptance, the sender transmits tokens in fragmented ESP-NOW frames. Since Cashu tokens may exceed the 250-byte ESP-NOW payload limit, tokens are fragmented:

```
Byte 0:       Message type (0x20 = TOKEN_FRAGMENT)
Bytes 1-2:    Token sequence number (uint16)
Byte 3:       Fragment index (uint8)
Byte 4:       Total fragments (uint8)
Bytes 5-249:  Token data (245 bytes per fragment)
```

4. The receiver reassembles fragments, verifies the complete token (same local verification as §3.4), and sends `RELAY_ACK` (0x21) for each successfully received token.
5. Upon receiving `RELAY_ACK`, the sender marks the token as relayed and deletes the local copy after a configurable confirmation window (default: 30 seconds).

Multi-hop relay: Each relayed token increments its `relay_count` and appends the relaying terminal's MAC to its `relay_path`. A maximum hop count (default: 10) prevents infinite relay loops. Terminals prioritize relaying to terminals advertising `has_internet=1`. When no internet-connected terminal is reachable, tokens relay toward terminals with lower hop counts to the nearest known internet-connected node, using a distance-vector approach:

- Each terminal maintains a `distance_to_internet` metric, initialized to 0 for internet-connected terminals and ∞ for offline terminals.
- Beacons include this metric. Terminals update their metric to `min(neighbor_distance + 1)` upon receiving neighbor beacons.
- Tokens flow "downhill" toward terminals with lower `distance_to_internet` values.

3.6.2 BLE Mesh Relay

BLE Mesh serves as a secondary relay mechanism for environments where ESP-NOW range is insufficient or where interoperability with non-ESP32 BLE Mesh devices is desired.

Token relay over BLE Mesh follows the Bluetooth SIG Mesh Profile with the following application-layer mapping:

- **Model:** A custom vendor model (Company ID: TBD) with two opcodes:
  - `TOKEN_RELAY_SEND` (opcode: 0xC0): Carries a Cashu token payload (segmented by the BLE Mesh transport layer for messages exceeding the Access Layer MTU).
  - `TOKEN_RELAY_ACK` (opcode: 0xC1): Acknowledges receipt.
- **Publishing:** Token relay messages are published to a designated group address. All terminals subscribe to this group.
- **TTL:** BLE Mesh message TTL is set to match the maximum desired hop count (default:

10).

3.7 Settlement Reconciliation

When a terminal regains internet connectivity, it executes the batch settlement process:

1. **Queue construction:** The terminal queries its local database for all unsettled tokens, ordered by TTL expiration (oldest first).
2. **Mint state check (NUT-07):** For each token, the terminal sends a `POST /v1/checkstate` request to the token's mint, providing the proof secrets (`Ys`). The mint responds with the state of each proof: `UNSPENT` , `PENDING` , or `SPENT` .
    - `UNSPENT` : Token is valid and unredeemed. Proceed to settlement.
    - `PENDING` : Token is being processed by another party. Retry after delay.
    - `SPENT` : Token has already been redeemed (double-spend detected). Mark as invalid, log incident, do not credit merchant.
3. **Redemption (NUT-05 — Melt):** For valid tokens, the terminal melts the proofs by having the mint pay a Lightning invoice:
    - The terminal generates a Lightning invoice (via BTCPay Server Greenfield API or directly via the merchant's Lightning node) for the total redeemable amount minus the mint's fee.
    - The terminal sends `POST /v1/melt/bolt11` to the mint with the invoice and proofs.
    - The mint pays the invoice and marks the proofs as spent.
    - The terminal marks the tokens as settled in local storage.
4. **Alternative: Swap and accumulate (NUT-03).** Instead of melting each token individually, the terminal may first swap all valid proofs for fresh proofs of consolidated denominations, then melt a single large token. This reduces the number of Lightning transactions required and amortizes routing fees.
5. **Settlement report:** The terminal generates a settlement report — a JSON document containing the number of tokens settled, total sats, any tokens that failed state checks (potential double-spends), and the relay paths of each token. This report is optionally published as a Nostr event (encrypted, kind 4 or 44) to the merchant's configured relay for record-keeping.

3.8 Security Analysis

Double-spend risk. The primary risk in offline ecash relay is double-spending: the

customer spends the same token at two different terminals before either can check with the mint. Mitigations:

- **Token TTL:** Shorter TTLs reduce the window in which a double-spend can go undetected. A 72-hour TTL bounds the maximum exposure.
- **Risk thresholds:** Terminals can enforce configurable maximum acceptance amounts for offline verification (e.g., accept at most 10,000 sats per token without online verification). Larger amounts require online state checking before confirmation.
- **Mint trust model:** Terminals only accept tokens from mints whose keysets they have cached. The merchant operator configures trusted mints during setup. Accepting tokens from untrusted mints is rejected.
- **Statistical detection:** Over time, the system logs double-spend incidents per mint and per payment source. Merchants can tighten acceptance rules for mints or token sizes with high double-spend rates.
- **Economic disincentive:** Cashu mints require Lightning payment to issue tokens. The cost of acquiring tokens for double-spending equals or exceeds the tokens' face value, limiting the economic incentive for attacks to the window of offline float.

Relay trust: Intermediate relay terminals handle tokens in transit. A malicious relay terminal could attempt to redeem tokens instead of forwarding them. Mitigation:

- **P2PK locking (NUT-10/NUT-11):** Tokens relayed through the mesh can be locked to the receiving merchant's public key. Only the intended recipient can redeem them at the mint. This requires the sender to know the recipient's public key, which is included in the ESP-NOW beacon.
- **Relay accountability:** The relay path (sequence of terminal MACs) is recorded. While MACs can be spoofed, the append-only path provides forensic data for dispute resolution within a trusted merchant network.

NFC security. NFC communication between the customer's phone and the terminal occurs at close range (~4 cm). Eavesdropping requires physical proximity. The token itself is a bearer instrument — possession equals ownership — so the NFC transfer is analogous to handing over physical cash.

3.9 Supply Chain Mesh Settlement

The preceding sections describe the flow of Cashu tokens from customer to merchant via NFC reception, local verification, mesh relay, and deferred Lightning settlement. This section extends the mesh-relayed ecash architecture to the second leg of the commercial supply chain: settlement from merchant to supplier. In conventional commerce, this leg is

mediated by wire transfers, trade credit, or card network payouts — all of which require persistent connectivity, intermediary trust, and multi-day settlement windows. The system described here enables a merchant terminal to accumulate customer-received Cashu tokens into supplier-designated pools, optionally rebalance their denominations, and forward them to supplier terminals either directly over the ESP-NOW/BLE mesh (for co-located suppliers) or via Lightning Network bridge (for remote suppliers), with a cryptographic audit trail linking every customer payment to its corresponding supplier settlement.

3.9.1 Token Accumulation Pool

Upon receiving and verifying a Cashu token from a customer (per §3.3 and §3.4), the merchant terminal stores the token in its local SQLite database. In addition to the primary token storage table described in §3.5, the terminal maintains a separate `supply_pool` table that tracks token allocation to configured suppliers:

```
CREATE TABLE supply_pool (
    pool_id         INTEGER PRIMARY KEY AUTOINCREMENT,
    supplier_npub   TEXT NOT NULL,
    token_data      TEXT NOT NULL,
    denomination    INTEGER NOT NULL,
    accumulated_sats INTEGER NOT NULL,
    created_at      INTEGER NOT NULL,
    forwarded_at    INTEGER DEFAULT NULL
);

CREATE INDEX idx_supply_pool_supplier ON supply_pool(supplier_npub);
CREATE INDEX idx_supply_pool_forwarded ON supply_pool(forwarded_at);
```

The `token_data` field stores the serialized Cashu token (NUT-00 cashuA or cashuB format). The `denomination` field records the individual proof denomination (per NUT-01 denomination set: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, … sats) of the largest proof in the token, enabling efficient denomination queries during rebalancing. The `accumulated_sats` field records the total satoshi value of all proofs in the token entry. The `forwarded_at` field remains NULL until the token is successfully forwarded to the supplier, at which point it is set to the Unix timestamp of the forwarding event.

Token allocation to suppliers is governed by the merchant's forwarding rules (§3.9.3). Upon each customer payment, the terminal inserts one or more rows into the `supply_pool` table according to the configured allocation — for example, allocating 60% of received tokens to a primary goods supplier and 40% to a secondary supplier. The allocation is performed by satoshi value: if a 500-sat customer payment is received and the merchant has configured a

60/40 split, the terminal selects proofs summing to approximately 300 sats for supplier A and 200 sats for supplier B, inserting separate pool entries for each. Because Cashu proof denominations are powers of two, exact splits are not always possible; the terminal rounds to the nearest achievable denomination boundary and accumulates residuals in a general-purpose pool.

3.9.2 Denomination Rebalancing

Customer payments arrive in small denominations. A 2,100-sat coffee payment may consist of proofs with denominations such as 1024 + 512 + 256 + 128 + 64 + 64 + 32 + 16 + 4 sats. When the merchant forwards accumulated tokens to a wholesale supplier — where invoices may total 50,000 sats or more — transmitting hundreds of small-denomination proofs is inefficient for both storage and mesh bandwidth.

The terminal performs denomination rebalancing before forwarding, contingent on connectivity state:

Online rebalancing (preferred). When the terminal has internet connectivity, it executes a NUT-03 swap against the mint to consolidate small-denomination proofs into larger ones. The swap request (per NUT-03 specification) takes the form:

```
POST /v1/swap
Content-Type: application/json

{
  "inputs": [
    {"amount": 1, "id": "00a1b2c3d4e5", "secret": "...", "C": "02..."},
    {"amount": 2, "id": "00a1b2c3d4e5", "secret": "...", "C": "02..."},
    {"amount": 4, "id": "00a1b2c3d4e5", "secret": "...", "C": "02..."},
    ...
  ],
  "outputs": [
    {"amount": 4096, "id": "00a1b2c3d4e5", "B_": "02..."},
    {"amount": 4096, "id": "00a1b2c3d4e5", "B_": "02..."},
    ...
  ]
}
```

The `inputs` array contains the existing small-denomination proofs. The `outputs` array contains blinded messages (`B_` values) for the desired larger denominations, where each `B_` is computed as `B_ = Y + r * G` (Y = hash_to_curve(new_secret), r = random blinding factor). The mint returns blind signatures (`C_` values) for each output, which the terminal unblinds to obtain fresh proofs of the consolidated denominations. The total input amount

must equal the total output amount (swap is value-preserving, excluding any mint fee defined in NUT-02 keyset metadata).

Alternative online path (melt and re-mint). When swapping is unavailable or the terminal wishes to change keysets, the terminal melts the small-denomination proofs via NUT-05 (having the mint pay a self-generated Lightning invoice) and immediately mints fresh large-denomination proofs via NUT-04 (paying the mint's Lightning invoice). This incurs Lightning routing fees but achieves the same result.

Offline forwarding (fallback). When the terminal lacks connectivity, denomination rebalancing is deferred. The terminal forwards the raw accumulated proofs to the supplier as-is. The supplier's terminal performs the swap when it next obtains connectivity. This is acceptable because Cashu tokens are bearer instruments — the supplier can redeem or swap them independently regardless of denomination composition.

3.9.3 Automated Forwarding Rules

The merchant configures per-supplier forwarding rules via the Plaza mobile application. Rules are stored as a JSON configuration object in the terminal's LittleFS filesystem at `/config/supplier_rules.json`:

```
{
  "supplier_rules": [
    {
      "supplier_npub": "npub1supplier1abc...",
      "supplier_terminal_mac": "AA:BB:CC:DD:EE:FF",
      "forward_threshold_sats": 50000,
      "forward_schedule_cron": "0 6 * * *",
      "forward_mode": "threshold",
      "settlement_path": "auto",
      "lightning_invoice_template": null,
      "allocation_pct": 60,
      "nostr_dm_request_invoice": true
    },
    {
      "supplier_npub": "npub1supplier2def...",
      "supplier_terminal_mac": null,
      "forward_threshold_sats": 100000,
      "forward_schedule_cron": null,
      "forward_mode": "manual",
      "settlement_path": "lightning",
      "lightning_invoice_template": "lnurl1...",
      "allocation_pct": 40,
      "nostr_dm_request_invoice": false
    }
```

```
    ]
  }
```

The `forward_mode` field accepts three values:

- **`threshold`** : The terminal evaluates the supplier's pool on each main loop iteration (approximately every 5 seconds). When the sum of `accumulated_sats` across all unforwarded rows (`forwarded_at IS NULL`) for the supplier's npub exceeds `forward_threshold_sats` , the terminal initiates forwarding.
- **`schedule`** : The terminal evaluates the cron expression in `forward_schedule_cron` against its local RTC clock. When the current time matches the cron schedule, the terminal forwards all accumulated tokens regardless of amount.
- **`manual`** : The terminal forwards only when the merchant explicitly triggers forwarding via the Plaza mobile application (which sends a BLE command to the terminal) or via a physical button press on the terminal hardware.

The `settlement_path` field determines the forwarding mechanism:

- **`mesh`** : Forward exclusively via ESP-NOW/BLE mesh (§3.9.4). Requires the supplier's terminal MAC.
- **`lightning`** : Forward exclusively via Lightning bridge (§3.9.5). Requires connectivity.
- **`auto`** : Attempt mesh forwarding first (if `supplier_terminal_mac` is non-null and the supplier's terminal is discoverable on the mesh); fall back to Lightning if mesh delivery fails or the supplier is unreachable after three retry attempts with exponential backoff (5s, 15s, 45s).

The `allocation_pct` field governs what percentage of incoming customer tokens are allocated to this supplier's pool. The sum of all `allocation_pct` values across configured suppliers must not exceed 100; any remainder is retained in the merchant's general revenue pool for withdrawal or discretionary spending.

3.9.4 Settlement Path A — Mesh-Local Supplier

When the supplier operates a Plaza terminal on the same ESP-NOW or BLE Mesh network — for example, a produce vendor and a restaurant operating adjacent stalls at a farmers market — the merchant terminal forwards accumulated tokens directly over the mesh without requiring internet connectivity.

This path uses a dedicated ESP-NOW frame type to distinguish supply chain settlements from consumer payment relays:

SUPPLY_CHAIN_PAYMENT frame (type `0x03`):

```
Byte 0:          Message type (0x03 = SUPPLY_CHAIN_PAYMENT)
Bytes 1-6:       Sender (merchant) terminal MAC
Bytes 7-12:      Recipient (supplier) terminal MAC
Bytes 13-44:     Supplier npub (32 bytes, x-only Schnorr public key)
Bytes 45-76:     Invoice reference (32 bytes, SHA-256 hash of Nostr DM order event
Byte 77:         Fragment index (uint8)
Byte 78:         Total fragments (uint8)
Bytes 79-80:     Total payment amount in sats (uint16; 0xFFFF = extended, see byte
Bytes 81-84:     Extended payment amount in sats (uint32, present only if bytes 79
Bytes 81/85-N:   Token payload (Cashu token data, up to 245 or 241 bytes per fragm
Bytes N+1-N+64: Merchant Schnorr signature (64 bytes, computed over SHA-256 of by
```

For token payloads exceeding the single-frame capacity (approximately 160 bytes after headers and signature), the terminal fragments the serialized Cashu token across multiple frames using the fragment index and total fragments fields, consistent with the fragmentation scheme described in §3.6 for consumer payment relay frames. The Schnorr signature is computed over the concatenation of all fragments' non-signature bytes and is included in the final fragment; intermediate fragments carry a 64-byte zero-padded signature field.

The Schnorr signature uses the merchant terminal's Nostr secret key (secp256k1) and follows BIP-340: `sig = schnorr_sign(merchant_nsec, SHA-256(frame_bytes_excluding_sig))`. The supplier's terminal verifies this signature against the merchant's known public key before accepting the tokens.

Acknowledgment frame (type `0x04`):

```
Byte 0:          Message type (0x04 = SUPPLY_CHAIN_ACK)
Bytes 1-6:       Sender (supplier) terminal MAC
Bytes 7-12:      Recipient (merchant) terminal MAC
Bytes 13-44:     SHA-256 hash of the received token payload
Bytes 45-48:     Acknowledged amount in sats (uint32)
Bytes 49-112:    Supplier Schnorr signature (64 bytes, over bytes 0-48)
Bytes 113-114:   Checksum (CRC-16)
```

Upon receiving the `SUPPLY_CHAIN_ACK`, the merchant terminal sets the `forwarded_at` timestamp on the corresponding `supply_pool` rows. If no acknowledgment is received within 30 seconds, the merchant terminal retransmits the `SUPPLY_CHAIN_PAYMENT` frame. After three failed attempts, the terminal marks the forwarding as failed and retries on the next loop iteration or falls back to Lightning (if `settlement_path` is `auto`).

This mechanism constitutes a mesh-native clearinghouse: value transfer between

commercial counterparties occurs over a local radio mesh without any intermediary, internet connection, or centralized settlement infrastructure. No analogous system exists in current point-of-sale or supply chain payment architectures.

3.9.5 Settlement Path B — Remote Supplier (Lightning Bridge)

When the supplier does not operate a terminal on the local mesh — for example, a wholesale distributor located in a different city — the merchant terminal settles via the Lightning Network when connectivity is available.

The settlement sequence proceeds as follows:

1. **Invoice acquisition.** The terminal obtains a Lightning invoice from the supplier via one of three mechanisms:
    - **Pre-configured LNURL:** The `lightning_invoice_template` field in the supplier rule contains an LNURL-pay endpoint. The terminal resolves the LNURL to obtain a fresh BOLT 11 invoice for the forwarding amount.
    - **Nostr DM request (NIP-44):** If `nostr_dm_request_invoice` is true, the terminal publishes an encrypted Nostr DM (NIP-44) to the supplier's npub requesting an invoice for the accumulated amount. The supplier's system (terminal, wallet, or automated responder) replies with a BOLT 11 invoice. The terminal monitors configured relays for the response.
    - **NWC (NIP-47):** If the merchant has configured a Nostr Wallet Connect string, the terminal can pay the supplier's Lightning address or BOLT 11 invoice directly via the NWC protocol, delegating invoice payment to the merchant's connected remote wallet.
2. **Token melt (NUT-05).** The terminal submits a melt request to the Cashu mint:

```
POST /v1/melt/bolt11
Content-Type: application/json

{
  "quote": "<melt_quote_id>",
  "inputs": [
    {"amount": 4096, "id": "00a1b2c3d4e5", "secret": "...", "C": "02..."},
    {"amount": 4096, "id": "00a1b2c3d4e5", "secret": "...", "C": "02..."},
    ...
  ]
}
```

Prior to submitting the melt, the terminal obtains a melt quote via `POST`

`/v1/melt/quote/bolt11` with the supplier's Lightning invoice, receiving a `quote` identifier and the mint's fee estimate. The `inputs` array contains proofs from the supplier's pool totaling the invoice amount plus the mint's fee.

3. **Settlement confirmation.** The mint pays the Lightning invoice and returns the payment preimage in the melt response. The terminal stores the preimage as proof of settlement and sets the `forwarded_at` timestamp on the corresponding `supply_pool` rows.
4. **Connectivity-triggered execution.** The terminal checks network connectivity on each main loop iteration. When the terminal transitions from offline to online and forwarding conditions are met (threshold exceeded or schedule matched), it executes the melt-and-pay sequence automatically. This allows tokens accumulated over hours or days of offline operation to settle in a single batch upon connectivity restoration.

3.9.6 Cryptographic Audit Trail

The merchant terminal acts as a payment intermediary: it receives tokens from customers and forwards value to suppliers. This intermediary role necessitates a verifiable audit trail linking inbound customer payments to outbound supplier settlements. The system constructs this trail using Schnorr-signed receipts anchored to the merchant's Nostr identity.

Customer receipt (generated at token reception, §3.3). When the terminal receives and verifies a Cashu token from a customer, it generates a signed receipt:

```
{
  "type": "customer_receipt",
  "version": 1,
  "customer_token_hash": "<SHA-256 hex digest of serialized Cashu token>",
  "amount_sats": 2100,
  "mint_url": "https://mint.example.com",
  "keyset_id": "00a1b2c3d4e5",
  "proof_count": 8,
  "timestamp": 1740249600,
  "terminal_npub": "npub1merchantterminal...",
  "signature": "<BIP-340 Schnorr signature over SHA-256(all fields except signatu
}
```

The receipt is stored in a local `receipts` table in the SQLite database. The `customer_token_hash` field is the SHA-256 digest of the complete serialized Cashu token (the same token stored in the primary token table per §3.5), serving as a cryptographic commitment to the exact tokens received. Optionally, the terminal publishes the receipt as a Nostr event. Event kinds 5763 (customer receipt) and 5764 (supplier payment proof) are

Plaza-specific application event kinds, not currently submitted as ratified NIPs. They are published here as an open specification; any implementation may use these kind numbers for interoperable supply chain audit events.

```
{
  "kind": 5763,
  "pubkey": "<terminal_npub_hex>",
  "created_at": 1740249600,
  "content": "<JSON-serialized customer_receipt>",
  "tags": [
    ["t", "plaza_receipt"],
    ["t", "customer_payment"],
    ["amount", "2100"]
  ],
  "sig": "<Nostr event signature>"
}
```

Supplier payment proof (generated at forwarding). When tokens are forwarded to a supplier via either settlement path, the terminal generates a signed payment proof:

```
{
  "type": "supplier_payment",
  "version": 1,
  "supplier_npub": "npub1supplier...",
  "amount_sats": 50000,
  "source_receipt_hashes": [
    "a3f2b8c1...",
    "d4e5f6a7...",
    "..."
  ],
  "token_hash": "<SHA-256 hex digest of forwarded Cashu tokens>",
  "settlement_path": "mesh",
  "lightning_preimage": null,
  "mesh_ack_hash": "<SHA-256 hex digest of SUPPLY_CHAIN_ACK frame>",
  "timestamp": 1740250000,
  "terminal_npub": "npub1merchantterminal...",
  "signature": "<BIP-340 Schnorr signature over SHA-256(all fields except signatu
}
```

The `source_receipt_hashes` array contains the `customer_token_hash` values from each customer receipt whose tokens contributed to this supplier payment, establishing a cryptographic link from individual customer transactions to the aggregate supplier settlement. For Lightning-settled payments, the `lightning_preimage` field contains the payment preimage (proof that the Lightning invoice was paid) and `mesh_ack_hash` is null.

For mesh-settled payments, `mesh_ack_hash` contains the SHA-256 hash of the supplier's `SUPPLY_CHAIN_ACK` frame and `lightning_preimage` is null.

The payment proof is optionally published as a Nostr event:

```
{
  "kind": 5764,
  "pubkey": "<terminal_npub_hex>",
  "created_at": 1740250000,
  "content": "<JSON-serialized supplier_payment>",
  "tags": [
    ["t", "plaza_receipt"],
    ["t", "supplier_payment"],
    ["p", "<supplier_npub_hex>"],
    ["amount", "50000"]
  ],
  "sig": "<Nostr event signature>"
}
```

End-to-end verification chain. An auditor (the merchant, the supplier, or any third party with access to the Nostr events) can independently verify the complete chain:

1. Retrieve all kind 5763 events from the merchant's npub. Verify each event's Nostr signature. Extract the `customer_token_hash` and `amount_sats` from each receipt.
2. Retrieve all kind 5764 events from the merchant's npub. Verify each event's Nostr signature. For each supplier payment, verify that every hash in `source_receipt_hashes` corresponds to a valid kind 5763 receipt.
3. Sum the `amount_sats` of the linked customer receipts and confirm consistency with the supplier payment's `amount_sats` (accounting for merchant margin).
4. Verify settlement finality: for Lightning-settled payments, the `lightning_preimage` can be validated against the payment hash of the supplier's invoice; for mesh-settled payments, the `mesh_ack_hash` can be verified against the supplier's corresponding acknowledgment record.

The entire chain is signed by the merchant's Nostr key (secp256k1 Schnorr), providing non-repudiable proof of the payment flow. Because Nostr events are distributed across relays and timestamped, this constitutes a decentralized, cryptographically verifiable supply chain settlement ledger.

3.9.7 Multi-Party Settlement Networks

The two-party merchant→supplier settlement described above extends naturally to N-party

supply chains. Consider a chain: Customer → Retailer → Distributor → Manufacturer. Each intermediary operates a Plaza terminal and maintains its own `supply_pool` table and forwarding rules:

- The retailer's terminal accumulates customer tokens and forwards to the distributor per §3.9.3–§3.9.5.
- The distributor's terminal accumulates tokens from multiple retailers and forwards to the manufacturer using the same mechanisms.
- Each intermediary generates signed receipts (kind 5763) and payment proofs (kind 5764), forming a chain of Nostr events that traces value flow from the original customer payment through each link of the supply chain.

For co-located supply chain partners — for example, a restaurant, a produce vendor, and a bakery supplier operating at the same farmers market — all settlement occurs over the local ESP-NOW mesh via `SUPPLY_CHAIN_PAYMENT` frames (§3.9.4). No internet connectivity is required at any point in the chain. Tokens flow upstream from customer to raw material supplier entirely over radio mesh, with each intermediary adding its signed audit record to the verifiable chain.

The `source_receipt_hashes` field in the supplier payment proof enables chain traversal: an auditor following the hash chain from a manufacturer's received payment can trace backward through each intermediary's kind 5764 events, through their linked kind 5763 receipts, to the original customer transactions. This provides end-to-end supply chain payment transparency without requiring any centralized ledger, payment processor, or trusted intermediary.

3.9.8 Security Considerations

Double-spend in the supply chain context. A malicious merchant could forward the same Cashu tokens to multiple suppliers. This is mitigated by the supplier terminal performing a NUT-07 state check (`POST /v1/checkstate`) when it next obtains internet connectivity. Tokens that have already been redeemed by another party return a `SPENT` state, alerting the supplier to the double-forward. In a mesh-only (fully offline) context, this check is deferred; the first supplier to reach connectivity and redeem the tokens receives the value. Suppliers operating in prolonged offline environments should negotiate shorter forwarding intervals or require P2PK-locked tokens (NUT-10/NUT-11) that can only be redeemed by the supplier's key.

Denomination rebalancing availability. NUT-03 swap and NUT-04/NUT-05 melt/mint operations require connectivity to the Cashu mint. In fully offline operation, tokens are forwarded with their original denomination composition. The supplier's terminal accepts the

denomination mismatch and performs rebalancing upon its own connectivity restoration.

Merchant signature verification. Supplier terminals must verify the BIP-340 Schnorr signature in the `SUPPLY_CHAIN_PAYMENT` frame before accepting tokens. The supplier's terminal must have the merchant's public key pre-registered (either configured manually or discovered via a prior Nostr event exchange). Frames with invalid or unrecognized signatures are rejected.

Lightning settlement security. When settling via Lightning bridge (§3.9.5), standard Lightning payment security applies: the payment preimage returned by the mint upon successful NUT-05 melt constitutes cryptographic proof that the supplier's invoice was paid. The preimage is stored in the audit trail and can be independently verified by hashing it (SHA-256) and comparing against the payment hash embedded in the BOLT 11 invoice.

This extension of the mesh-relayed ecash architecture from consumer→merchant payments to merchant→supplier settlement transforms Innovation I from a point-of-sale system into a multi-party commerce network. The combination of mesh-local token forwarding, automated Lightning bridging, and Nostr-anchored cryptographic audit trails enables complete supply chain settlement — from end consumer to raw material supplier — using only open protocols, bearer ecash instruments, and local radio mesh, with no dependence on banking infrastructure, payment processors, or persistent internet connectivity.

---

4. Innovation II: Beacon-Driven Commerce Discovery Pipeline

4.1 Overview

This section describes a system in which Plaza merchant terminals continuously broadcast BLE beacon advertisements containing structured merchant metadata, enabling customer smartphones to passively discover nearby merchants, browse their product offerings, and complete purchases via NFC Cashu token transfer — all without internet connectivity on either device.

4.2 BLE Beacon Advertisement Format

Each Plaza terminal broadcasts BLE advertisements in two formats for maximum device compatibility:

4.2.1 iBeacon Format

The iBeacon advertisement uses Apple's iBeacon specification:

- **UUID:** A Plaza-specific UUID (e.g., `A495FF00-C5B1-4B44-B512-1370F02D74DE`)

identifying the advertisement as a Plaza commerce beacon.

- **Major:** Encoded merchant category (16-bit):
  - `0x0001` : Food & Beverage
  - `0x0002` : Retail
  - `0x0003` : Services
  - `0x0004` : Market Vendor
  - `0x0005` : Vending Machine
  - (additional categories defined in specification)
- **Minor:** Encoded subcategory and flags (16-bit):
  - Bits 15-12: Subcategory within major category
  - Bit 11: Cashu accepted (1=yes)
  - Bit 10: Lightning accepted (1=yes)
  - Bit 9: Mesh relay available (1=yes)
  - Bits 8-0: Price tier indicator (0-511, mapping to price range)

4.2.2 Eddystone-URL Format

The Eddystone-URL frame broadcasts a compressed URL pointing to the merchant's Nostr profile or product catalog:

- **URL Scheme:** `https://` (prefix byte 0x03)
- **Encoded URL:** Up to 17 bytes of URL-encoded merchant identifier, e.g., `plazabtc.com/m/` followed by a short merchant ID.

4.2.3 Extended Beacon Data via Scan Response

Since iBeacon and Eddystone-URL frames have limited payload capacity, the terminal includes extended merchant metadata in the BLE scan response packet. The scan response uses a Manufacturer Specific Data field (Company ID: TBD, pending Bluetooth SIG registration; prototype uses 0xFFFF) with the following structure:

```
Byte 0:       Plaza protocol version (0x01)
Byte 1:       Data type (0x01 = Merchant Info)
Bytes 2-3:    Merchant name length (uint16, little-endian)
```

```
Bytes 4-N:    Merchant name (UTF-8, max 32 bytes)
Bytes N+1-N+2: Product count (uint16)
Bytes N+3-N+4: Mint URL length (uint16)
Bytes N+5-M:  Cashu mint URL (UTF-8, max 64 bytes)
Bytes M+1-M+4: Geohash (32-bit truncated geohash for location)
Bytes M+5-M+6: Checksum (CRC-16)
```

4.2.4 Product Catalog Beacon

In addition to the primary beacon, terminals cycle through secondary beacon frames advertising individual products. Each product frame uses the Manufacturer Specific Data field:

```
Byte 0:       Plaza protocol version (0x01)
Byte 1:       Data type (0x02 = Product Listing)
Bytes 2-3:    Product index (uint16)
Bytes 4-5:    Price in sats (uint16, for prices ≤65535 sats; 0xFFFF = extended pr
Bytes 6-7:    Extended price (uint32, present only if bytes 4-5 == 0xFFFF)
Bytes 6/8-N:  Product name (UTF-8, max 40 bytes)
Byte N+1:     Availability (0x00=out of stock, 0x01=available, 0x02=limited)
Bytes N+2-N+3: Checksum (CRC-16)
```

Terminals cycle through product advertisements at a configurable rate (default: one product per beacon interval, 100ms interval, cycling through the full catalog). A terminal with 10 products completes a full catalog broadcast every 1 second.

4.3 Customer Discovery Flow

The complete discovery-to-payment pipeline operates as follows:

Step 1: Passive Scanning. The customer's smartphone runs the Plaza app (or any BLE-scanning application aware of the Plaza beacon format). The app performs BLE scanning in the background, filtering for the Plaza iBeacon UUID. No internet connectivity is required for scanning.

Step 2: Merchant List Assembly. As beacons are received, the app assembles a list of nearby merchants with:

- Merchant name (from scan response)
- Category (from iBeacon Major/Minor)
- Distance estimate (from RSSI)
- Payment capabilities (Cashu/Lightning/Mesh flags)

- Cashu mint URL (from scan response)

Step 3: Product Catalog Assembly. Upon selecting a merchant, the app begins capturing the product catalog beacon frames from that specific terminal (filtered by the terminal's BLE MAC address). The app accumulates product listings until the full catalog is received (determined by product count from the merchant info frame matching received unique product indices).

Step 4: Product Selection. The customer browses the locally assembled catalog and selects a product. The app displays the price in sats and prepares payment.

Step 5: Cashu Token Preparation. The app checks the customer's local Cashu wallet for proofs issued by the mint URL broadcast in the merchant's beacon. If sufficient proofs are available:

- The app constructs a Cashu token (NUT-00) with proofs summing to the product price.
- The token is serialized and written to an NDEF record on the phone's NFC interface.

If the customer lacks proofs from the required mint but has proofs from a different mint:

- If online, the app can swap between mints (melt at old mint, mint at new mint).
- If offline, the app displays an error suggesting the customer obtain tokens from the advertised mint.

Step 6: NFC Payment. The customer taps their phone against the terminal's PN532 NFC reader. The terminal reads the NDEF record, parses the Cashu token, performs local verification (as described in §3.4), and confirms the transaction on its display.

Step 7: Receipt and Social Signal. Upon confirmation:

- The terminal optionally writes an NDEF receipt back to the phone via NFC, containing: transaction ID, timestamp, merchant name, product name, amount, and a Nostr event ID for future reference.
- When online, the customer's app can publish an optional social signal (a Nostr note) indicating the purchase, contributing to the merchant's discoverable reputation.

4.4 Geohash-Tagged Nostr Events for Online Discovery

The offline BLE discovery system is complemented by online discovery via Nostr:

Each Plaza merchant terminal publishes a merchant location event to configured Nostr relays when online:

```
{
```

```
  "kind": 38383,
  "pubkey": "<merchant_terminal_pubkey>",
  "created_at": 1740249600,
  "content": "Next Block Coffee - Specialty coffee, pastries, Bitcoin merch",
  "tags": [
    ["d", "<unique_merchant_identifier>"],
    ["g", "9q8yyk8y"],
    ["g", "9q8yyk8"],
    ["g", "9q8yyk"],
    ["g", "9q8yy"],
    ["t", "coffee"],
    ["t", "food"],
    ["t", "bakery"],
    ["cashu_mint", "https://mint.example.com"],
    ["lnaddress", "merchant@plazabtc.com"],
    ["hours", "Mon-Fri 07:00-17:00 PST"],
    ["nfc", "cashu,lightning"],
    ["mesh", "espnow,ble"],
    ["plaza_version", "1.0"],
    ["name", "Next Block Coffee"],
    ["image", "<url_to_merchant_image>"]
  ],
  "sig": "<schnorr_signature>"
}
```

Multiple geohash prefixes of decreasing precision are included in the `g` tags to enable efficient geographic queries at different zoom levels. Client applications query relays with `{"kinds":[38383], "#g":["9q8yy"]}` to find merchants in a geographic area.

4.5 Offline-Online Continuity

The system provides seamless transition between offline and online operation:

- **Fully offline:** BLE beacons → product catalog → NFC Cashu payment. No internet on either device.
- **Customer online, merchant offline:** Customer discovers merchant via Nostr location events (online map), navigates to location, completes payment via NFC Cashu (offline).
- **Both online:** Customer discovers merchant via Nostr events or BLE beacon, pays via Lightning QR, NWC (NIP-47), or NFC Cashu.
- **Mesh-assisted:** If the merchant terminal is offline but within mesh range of an online terminal, the payment token can relay through the mesh for faster settlement (per Innovation I).

---

5. Innovation III: Agent-Native Vending on Nostr

5.1 Overview

This section describes a vending machine architecture in which the machine operates as an autonomous first-class participant on the Nostr protocol — not merely a device controlled by a remote application, but a social node with its own cryptographic identity, capable of publishing events, receiving messages, processing payments, and reporting its status through the same open protocol used by human participants.

5.2 The Sovereignty Stack

The agent-native vending machine embodies a complete sovereignty stack:

1. **Money:** Bitcoin (Lightning + Cashu). The machine holds its own wallet and processes payments directly. No payment intermediary.
2. **Compute:** Autonomous agent logic running on an ESP32 (or Raspberry Pi for prototypes). The machine makes decisions — accepting orders, dispensing products, managing inventory — without human intervention for routine operations.
3. **Communication:** Nostr protocol. The machine communicates with customers, its owner, and the broader network via signed Nostr events distributed through relays.
4. **Identity:** Nostr keypair (npub/nsec). The machine IS a participant on the network — discoverable, addressable, and verifiable by its public key.
5. **Commerce:** Vending operations. The machine publishes products, accepts orders, and completes sales autonomously.

Each layer is sovereign: the money is self-custodied, the compute is local, the communication is censorship-resistant, the identity is self-owned, and the commerce is peer-to-peer. No platform, corporation, or intermediary is required for any layer.

5.3 Nostr Identity and Event Architecture

5.3.1 Machine Keypair

During initial provisioning, the vending machine generates or is assigned a Nostr keypair (secp256k1 Schnorr). The secret key (nsec) is stored in the ESP32's encrypted NVS (Non-Volatile Storage) partition. The public key (npub) becomes the machine's permanent identity on the network.

The machine's Nostr profile (kind 0) is published to configured relays:

```
{
  "kind": 0,
  "pubkey": "<machine_npub_hex>",
  "content": "{\"name\":\"Plaza Vending #001\",\"about\":\"Autonomous Bitcoin ven
  "tags": [
    ["t", "vending"],
    ["t", "plaza"],
    ["t", "bitcoin"],
    ["L", "com.plazabtc.device"],
    ["l", "vending-machine", "com.plazabtc.device"]
  ]
}
```

5.3.2 Product Listings (NIP-15)

The machine publishes its inventory as NIP-15 Nostr Marketplace events.

Stall event (kind 30017):

```
{
  "kind": 30017,
  "pubkey": "<machine_npub_hex>",
  "content": "{\"name\":\"Plaza Vending #001\",\"description\":\"Self-service Bit
  "tags": [
    ["d", "plaza-vend-001-stall"],
    ["g", "9q8yyk"],
    ["t", "vending"]
  ]
}
```

Product events (kind 30018):

```
{
  "kind": 30018,
  "pubkey": "<machine_npub_hex>",
  "content": "{\"id\":\"espresso-001\",\"stall_id\":\"plaza-vend-001-stall\",\"na
  "tags": [
    ["d", "plaza-vend-001-espresso"],
    ["t", "coffee"],
    ["t", "espresso"],
    ["g", "9q8yyk"]
  ]
}
```

The `quantity` field reflects current inventory. The machine republishes product events

whenever inventory changes, providing real-time availability to any Nostr client querying its stall.

5.3.3 Order Flow (NIP-69)

Customers can order from the machine via Nostr. Note: NIP-69 is a draft proposal as of February 2026; the event kind numbers used below (5760–5762) are subject to change upon finalization of the NIP.

1. **Customer sends order** (NIP-69 order event, kind 5760, encrypted to the machine's npub):

```
{
  "kind": 5760,
  "pubkey": "<customer_npub_hex>",
  "content": "<encrypted: {\"items\":[{\"product_id\":\"espresso-001\",\"quant
  "tags": [
    ["p", "<machine_npub_hex>"]
  ]
}
```

2. **Machine responds with payment request** (kind 5761, encrypted to customer):

```
{
  "kind": 5761,
  "content": "<encrypted: {\"payment_options\":[{\"type\":\"ln\",\"link\":\"lr
  "tags": [
    ["p", "<customer_npub_hex>"],
    ["e", "<order_event_id>"]
  ]
}
```

3. **Payment completion:** The customer pays via Lightning (scanning the invoice QR), Cashu NFC tap (presenting tokens at the terminal's NFC reader), or Nostr zap (kind 9735 zap event to the machine's npub for the invoice amount).

4. **Machine confirms and dispenses** (kind 5762, encrypted to customer):

```
{
  "kind": 5762,
  "content": "<encrypted: {\"paid\":true,\"shipped\":true,\"message\":\"Espres
  "tags": [
    ["p", "<customer_npub_hex>"],
    ["e", "<order_event_id>"]
```

```
      ]
    }
```

5.3.4 Status and Reporting Events

The machine publishes operational status as Nostr events:

Inventory alert (kind 1 — standard note, from machine's npub):

```
Slot A1 (Espresso): 3 remaining ⚠️
Slot A2 (Cold Brew): EMPTY 🔴
Slot B1 (Bitcoin Sticker Pack): 22 remaining ✅
```

Revenue report (kind 4 or 44 — encrypted DM to owner's npub):

```
{
  "period": "2026-02-22",
  "transactions": 47,
  "revenue_sats": 98700,
  "by_product": {
    "espresso-001": {"qty": 23, "sats": 48300},
    "coldbrew-001": {"qty": 12, "sats": 25200},
    "sticker-001": {"qty": 12, "sats": 25200}
  },
  "by_method": {
    "cashu_nfc": 31,
    "lightning_qr": 14,
    "nostr_zap": 2
  },
  "uptime_pct": 99.7,
  "errors": []
}
```

Maintenance alert (encrypted DM to owner):

```
⚠️ Motor A2 stall detected. Slot A2 dispensing disabled. Error code: MOTOR_STALL_
```

5.4 Payment Processing

The vending machine accepts three payment methods:

1. **Cashu NFC tap (offline capable):** Customer taps phone with Cashu tokens against the PN532 NFC reader. Terminal verifies locally per §3.4 and §4.3. This is the primary offline payment method.

2. **Lightning QR (online required):** The machine displays a BOLT 11 Lightning invoice QR code on its ILI9341 display. The customer scans with any Lightning wallet. The machine monitors invoice payment status via its Lightning node connection (BTCPay Greenfield API or direct LND/CLN/Phoenixd API) and dispenses upon confirmation.
3. **Nostr zap (online required):** The customer sends a zap (NIP-57) to the machine's npub for the exact invoice amount. The machine monitors for incoming zap receipts on its configured relays, verifies the zap amount matches the outstanding order, and dispenses.

Payment priority: When the machine is offline, only Cashu NFC is available, and the machine's display indicates this. When online, all three methods are presented simultaneously (QR code on display for Lightning, NFC reader active for Cashu, Nostr listener active for zaps), and the first completed payment triggers dispensing.

5.5 Hardware Specification

The vending controller extends the Plaza Terminal hardware:

Component	Part	Function
Microcontroller	ESP32-S3 Feather (8MB flash, 2MB PSRAM)	Main processor, WiFi, BLE, ESP-NOW
NFC Reader	PN532 (Adafruit breakout)	Cashu token reception via NFC (ISO 14443A, NDEF)
Display	ILI9341 2.8″ or 3.5″ TFT (320×240 or 480×320)	QR codes, product selection, status display
Touch Panel	Resistive touch overlay (XPT2046 controller)	Product selection input
Relay Module	4-channel or 8-channel optoisolated relay board	Controls dispensing motors/solenoids
Motor Driver	L298N dual H-bridge (or individual MOSFETs per slot)	Drives DC motors for product dispensing
Dispensing Motors	DC gear motors (12V, one per slot)	Rotates coils/pushers to release products
Power Supply	12V 5A switching adapter	Powers motors; 3.3V regulator for ESP32
	Weatherproof steel or 3D-printed	

Enclosure	housing	Physical security and durability
Speaker (optional)	Piezo buzzer or small speaker + I2S DAC	Audio confirmation of payment
Camera (optional)	OV2640 module	Anti-vandalism monitoring

Dispensing sequence:

1. Payment verified (Cashu, Lightning, or zap).
2. ESP32 activates the relay corresponding to the ordered product's slot.
3. Relay energizes the dispensing motor for a calibrated duration (typically 1.5–3 seconds depending on motor and product).
4. Motor sensor (optical encoder or limit switch) confirms product fell to the pickup area.
5. If confirmation is not received within timeout (5 seconds), the motor reverses briefly and retries once. If retry fails, the machine generates a maintenance alert, refunds the customer (Cashu mint or Lightning), and disables the slot.

5.6 Remote Management via Nostr

The machine owner manages the vending machine entirely through Nostr — no proprietary cloud dashboard, no vendor-controlled management portal.

Owner commands (encrypted DMs to machine's npub):

Command	Description
`INVENTORY_UPDATE {"slot":"A1","qty":25}`	Set inventory count for a slot
`PRICE_UPDATE {"product":"espresso-001","price":2500}`	Update product price
`SLOT_ENABLE {"slot":"A2"}`	Re-enable a disabled slot
`SLOT_DISABLE {"slot":"B2","reason":"restocking"}`	Manually disable a slot
`STATUS_REQUEST`	Request immediate status report
`REBOOT`	Remote restart of the machine

`WITHDRAW {"amount":50000,"invoice":"lnbc..."}`	Withdraw accumulated revenue to owner's wallet

The machine authenticates owner commands by verifying the Nostr event signature against the owner's public key, which is configured during provisioning and stored in NVS. Only events signed by the registered owner npub are accepted as management commands.

5.7 Mesh Participation

The vending machine participates in the Plaza BLE mesh network (Innovation I) as a full node:

- It broadcasts BLE beacons advertising its products (Innovation II).
- It relays Cashu tokens received from other terminals via ESP-NOW.
- Given that vending machines are typically deployed in fixed locations with reliable power, they serve as high-uptime mesh relay nodes, anchoring the network.
- A vending machine with internet connectivity (e.g., via Ethernet or cellular backhaul) serves as a settlement gateway for the mesh — receiving relayed tokens from offline terminals and settling them to Lightning.

---

6. Security Considerations

6.1 Double-Spend Prevention in Offline Ecash

As discussed in §3.8, offline Cashu token acceptance carries inherent double-spend risk. The system employs a layered mitigation strategy:

1. **Configurable offline acceptance limits.** Merchants set a maximum per-transaction and per-hour amount for offline Cashu acceptance. Amounts exceeding the threshold require online state checking (NUT-07).
2. **Token TTL enforcement.** Tokens held beyond their TTL (default 72 hours) are prioritized for settlement and rejected for further relay.
3. **Mint trust lists.** Terminals only accept tokens from explicitly configured mints, limiting exposure to untrusted issuers.
4. **P2PK token locking.** For high-value offline transactions, the customer can lock the token to the merchant's public key (NUT-10/NUT-11), ensuring only the merchant can redeem it.

5. **Fraud logging and reporting.** All double-spend detections during settlement are logged and reported to the merchant. Persistent patterns from specific mints trigger automatic removal from the trust list.

6.2 Mesh Trust Model

The mesh relay network operates within a partially trusted model:

- **Enrollment:** Terminals join the mesh by exchanging ESP-NOW peer credentials. The merchant operator explicitly adds authorized peer terminals during setup.
- **Open relay vs. closed relay:** The system supports both modes. In closed relay mode, only enrolled peers relay tokens. In open relay mode, any Plaza terminal within range can relay. Open relay increases network resilience but expands the trust surface.
- **Token integrity:** Cashu token cryptographic properties (blind signatures) are preserved regardless of relay path. Relay nodes cannot modify tokens without invalidating them. Relay nodes can only steal tokens that are not P2PK-locked to a specific recipient.
- **Relay rate limiting:** Terminals enforce per-peer rate limits on relay requests to prevent flooding (default: 100 tokens per peer per hour).

6.3 NFC Security

- **Range limitation:** NFC operates at ~4 cm, limiting eavesdropping to physical proximity.
- **Bearer token model:** Cashu tokens are bearer instruments. NFC transfer is analogous to handing over physical cash. No additional authentication is required beyond possession.
- **NDEF integrity:** The token payload in the NDEF record is self-authenticating (Cashu blind signatures). Tampering with the NDEF record produces an invalid token that fails verification.

6.4 Nostr Key Management for Devices

- **Key storage:** Device secret keys are stored in the ESP32's encrypted NVS partition. The NVS encryption key is derived from the ESP32's eFuse-based flash encryption key, providing hardware-backed key protection.
- **Key rotation:** The system supports key rotation via the owner publishing a kind 1040 (key rotation) event signed by the old key, designating the new key. Connected clients and relays update their records.
- **Key compromise response:** If a device key is suspected compromised, the owner publishes a revocation event. Relays that support NIP-40 (event expiration) or custom

revocation kinds remove the compromised key's events.

- **Seed backup:** The device keypair is derived from a BIP-39 mnemonic seed generated during provisioning. The owner records this seed for recovery. A replacement device can restore the same identity from the seed.

---

7. Implementation Notes

7.1 Hardware

All three innovations share a common hardware platform:

- **Microcontroller:** ESP32-S3 (Adafruit Feather ESP32-S3 or equivalent). Dual-core Xtensa LX7 @ 240 MHz, 8 MB flash, 2 MB PSRAM, integrated WiFi 802.11 b/g/n, Bluetooth 5.0 LE, hardware AES-256/SHA acceleration.
- **NFC:** PN532 NFC/RFID controller (Adafruit breakout board). Supports ISO 14443A/B, NDEF read/write, SPI and I2C interfaces. Connected to ESP32 via SPI bus for maximum throughput.
- **Display:** ILI9341-based 2.8″ TFT (320×240) with resistive touch (XPT2046). Connected via SPI bus. Displays QR codes (generated using qrcode library compiled for ESP32), product catalogs, and transaction confirmations.
- **Storage:** ESP32's 8 MB flash partitioned: 1 MB application, 1 MB OTA update, 4 MB LittleFS filesystem (token storage, keyset cache, configuration), 2 MB SPIFFS (display assets).

7.2 Software Stack

- **Firmware:** Arduino framework (C++) for ESP32. Libraries: Adafruit PN532 (NFC), Adafruit ILI9341 (display), WiFi, esp_now (mesh), BLE (Bluetooth mesh), ArduinoJson (JSON parsing), mbedtls (cryptography — secp256k1, SHA-256, AES).
- **Nostr client:** Minimal Nostr client implemented in C++ on ESP32, supporting: event creation and signing (kind 0, 1, 4, 44, 30017, 30018, 5760-5762, 5763, 5764, 38383), WebSocket relay connection (when online), local event queue (when offline, events published upon connectivity restoration).
- **Cashu library:** Minimal Cashu token parser and verifier in C++, implementing: NUT-00 (deserialization), NUT-01/02 (keyset validation), NUT-03 (swap, online; deferred offline), NUT-05 (melt, online only), NUT-07 (state check, online only), NUT-10/11 (P2PK locking), NUT-12 (DLEQ verification). Blind signature verification uses mbedtls

secp256k1 operations.

- **Mobile application:** React Native (TypeScript) for iOS and Android. Libraries: react-native-nfc-manager (NFC), react-native-ble-plx (BLE scanning), nostr-tools (Nostr events), cashu-ts (Cashu wallet), react-native-maps or MapLibre (map view).

7.3 Nostr NIPs Used

NIP	Application
NIP-01	Basic event structure, signing, relay communication
NIP-02	Contact list (machine follows its owner; owner follows machine)
NIP-04	Legacy encrypted DMs (owner commands, revenue reports)
NIP-05	NIP-05 verification for machine identity
NIP-15	Marketplace product listings (stalls, products)
NIP-44	Versioned encrypted DMs (preferred over NIP-04)
NIP-47	Nostr Wallet Connect (customer wallet integration)
NIP-57	Zaps (payment method for vending)
NIP-59	Gift wrap (sealed sender for order privacy)
NIP-60	Cashu wallet state on Nostr
NIP-61	Nutzaps (ecash micropayment tips)
NIP-69	Peer-to-peer order events (draft NIP)
NIP-87	Mint discovery on Nostr
NIP-99	Classified listings
Kind 5763	Customer payment receipt (Plaza-specific, not a ratified NIP)
Kind 5764	Supplier payment proof (Plaza-specific, not a ratified NIP)

7.4 Cashu Protocol Integration

- **Token format:** NUT-00 cashuA and cashuB tokens.
- **Supported mints:** Any Cashu mint implementing NUT-00 through NUT-07. Mint URLs

and keysets cached locally on each terminal.

- **Mint discovery:** Via NIP-87 Nostr events and via BLE beacon advertisements (§4.2.3).
- **Settlement:** NUT-05 (melt) for redeeming tokens to Lightning. NUT-03 (swap) for consolidating proofs before melt.
- **Offline verification:** Keyset-based signature checking and optional DLEQ proof verification (NUT-12). See §3.4.
- **P2PK locking:** NUT-10/NUT-11 used for supply chain pool tokens and high-value relay tokens to enforce recipient-only redemption.

---

8. Conclusion

This paper has presented three novel technical innovations for decentralized Bitcoin commerce — the first of which encompasses two distinct contributions:

1. **Mesh-Relayed Ecash Commerce and Supply Chain Settlement** — a system for relaying Cashu bearer tokens between merchant terminals via ESP-NOW and BLE Mesh, enabling multi-hop offline payment settlement with deferred Lightning reconciliation. This innovation encompasses two distinct contributions: (a) the consumer→merchant payment relay (§3.1–§3.8), which allows commerce to continue functioning in the complete absence of internet connectivity with cryptographic guarantees inherited from the Cashu blind signature scheme; and (b) the merchant→supplier supply chain extension (§3.9), which enables automated token accumulation, denomination rebalancing, and upstream forwarding via either mesh-local relay or Lightning bridge, with a cryptographic audit trail using Schnorr-signed receipts (Nostr event kinds 5763 and 5764) linked by hash chains to provide end-to-end supply chain payment transparency without centralized ledgers.
2. **Beacon-Driven Commerce Discovery Pipeline** — a combined BLE beacon advertising and NFC payment system that enables the complete discovery-to-payment flow (merchant discovery → product browsing → product selection → payment completion) without internet connectivity on either the customer or merchant device. This system bridges online Nostr-based merchant discovery with offline BLE-based local discovery through a unified protocol.
3. **Agent-Native Vending on Nostr** — a vending machine architecture in which the machine operates as a sovereign node on the Nostr network with its own cryptographic identity, publishing inventory, accepting orders, processing payments (Cashu NFC, Lightning, Nostr zaps), and communicating with its owner — all through the open Nostr

protocol without proprietary infrastructure.

All three innovations are built entirely on open protocols (Nostr, Lightning Network, Cashu, ESP-NOW, BLE Mesh, NFC/NDEF) and open hardware (ESP32). The author commits to open-source release of all implementations under permissive licensing.

This paper is published as a defensive disclosure. The technical details contained herein are placed in the public domain for the purpose of establishing prior art, ensuring that these methods remain available for anyone to implement and that no party may obtain exclusive patent rights over them.

References

[1] Cashu Protocol Specification. "Cashu NUTs (Notation, Usage & Terminology)." https://github.com/cashubtc/nuts. Accessed February 2026.

[2] Chaum, D. "Blind Signatures for Untraceable Payments." Advances in Cryptology — CRYPTO '82, Springer, 1983.

[3] Espressif Systems. "ESP-NOW User Guide." https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html. Accessed February 2026.

[4] Espressif Systems. "ESP32-S3 Series Datasheet." https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf. Accessed February 2026.

[5] Bluetooth SIG. "Mesh Profile Specification v1.0.1." https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/. Accessed February 2026.

[6] Fiatjaf et al. "Nostr Protocol." https://github.com/nostr-protocol/nostr. Accessed February 2026.

[7] NIP-01: Basic Protocol Flow Description. https://github.com/nostr-protocol/nips/blob/master/01.md.

[8] NIP-15: Nostr Marketplace. https://github.com/nostr-protocol/nips/blob/master/15.md.

[9] NIP-47: Nostr Wallet Connect. https://github.com/nostr-protocol/nips/blob/master/47.md.

[10] NIP-60: Cashu Wallet. https://github.com/nostr-protocol/nips/blob/master/60.md.

[11] NIP-61: Nutzaps. https://github.com/nostr-protocol/nips/blob/master/61.md.

[12] NIP-69: Peer-to-Peer Order Events (draft). https://github.com/nostr-protocol/nips/blob/master/69.md.

[13] NIP-87: Relay Management API — Mint Discovery. https://github.com/nostr-protocol/nips/blob/master/87.md.

[14] NIP-99: Classified Listings. https://github.com/nostr-protocol/nips/blob/master/99.md.

[15] Poon, J. and Dryja, T. "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments." 2016. https://lightning.network/lightning-network-paper.pdf.

[16] BTCPay Server. "Greenfield API Documentation." https://docs.btcpayserver.org/API/Greenfield/v1/. Accessed February 2026.

[17] Dorsey, J. Bitchat — Bluetooth mesh chat. https://github.com/jackjackbits/bitchat. Accessed February 2026.

[18] Apple Inc. "iBeacon." https://developer.apple.com/ibeacon/. Accessed February 2026.

[19] Google. "Eddystone Protocol Specification." https://github.com/google/eddystone. Accessed February 2026.

[20] NFC Forum. "NFC Data Exchange Format (NDEF) Technical Specification." https://nfc-forum.org/. Accessed February 2026.

This document is published as a defensive disclosure under the principles of statutory invention registration and defensive publication. The innovations described herein are disclosed to the public to serve as prior art against future patent applications covering the same or substantially similar methods. Published February 22, 2026, by Alejandro Diaz, Plaza (plazabtc.com), Oakland, California.